**Один день из жизни DevOps**

Устанавливаем Убунту.

Создаем каталог Hello внутри домашнего каталога пользователя:

**mkdir hello**

**cd hello**

**ls**

Создаем веб приложение внутри каталога hello:

user@Ubuntu:~/hello$ **nano app.py**

*#!/usr/bin/env python3*

*import datetime*

*def do\_magic():*

*now = datetime.datetime.now()*

*return "Hello! {0}".format(now)*

*if \_\_name\_\_ == "\_\_main\_\_":*

*print(do\_magic())*

user@Ubuntu:~/hello$ **chmod +x ./app.py**

user@Ubuntu:~/hello$ **./app.py**

Ставим Гит для хранения исходных текстов (контроль версий). Что такое Гит – эта система управления исходным кодом. По большей части это файл который мы запускаем к которому мы передаем команды что с репозиторием сделать.

user@Ubuntu:~/hello$ **sudo apt install git**

user@Ubuntu:~/hello$ **git status**

fatal: not a git repository (or any of the parent directories): .git

Видим что нет репозитория.

Создаем репозиторий:

user@Ubuntu:~/hello$ **git init**

hint: Using 'master' as the name for the initial branch. This default branch name

hint: is subject to change. To configure the initial branch name to use in all

hint: of your new repositories, which will suppress this warning, call:

hint:

hint: git config --global init.defaultBranch <name>

hint:

hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and

hint: 'development'. The just-created branch can be renamed via this command:

hint:

hint: git branch -m <name>

Initialized empty Git repository in /home/user/hello/.git/

Проверяем каталог что появился скрытый каталог .git, эта папочка говорит что считает ли Гит этот каталог репозиторием:

user@Ubuntu:~/hello$ **ls -la**

total 16

drwxrwxr-x 3 user user 4096 Oct 15 12:22 .

drwxr-x--- 6 user user 4096 Oct 15 12:22 ..

-rwxrwxr-x 1 user user 169 Oct 15 12:15 **app.py**

drwxrwxr-x 7 user user 4096 Oct 15 12:22 **.git**

Проверяем и видим, что нету комитов (сохранения исходного кода):

On branch master

**No commits yet**

Untracked files:

(use "git add <file>..." to include in what will be committed)

app.py

nothing added to commit but untracked files present (use "git add" to track)

user@Ubuntu:~/hello$ **git status**

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

app.py

nothing added to commit but untracked files present (use "git add" to track)

По умолчанию после создания репозитория в каталоге, гит не считает файлы в нем как исходный код (untracked – не следит), для того чтобы он начал принимать код, нужно объяснить ему кто будет добавлять код, а именно создать пользователя. Это важно, потому-что когда вы сохраняете некую версию исходного кода, вы делаете комит, тем самым вы должны написать кто сделал этот комит – как его зовут и какой у него адрес электронной почты. Если сделать без этих настроек, то гит не даст добавить код.

Поэтому добавляем пользователя:

user@Ubuntu:~/hello$ **git config --global user.email** [**alm15@mail.ru**](mailto:alm15@mail.ru)

user@Ubuntu:~/hello$ **git config --global user.name “Almaskhan Oskenbayev”**

user@Ubuntu:~/hello$ **git add app.py**

user@Ubuntu:~/hello$ **git commit -m 'first version'’**

Узнать кто вносил изменения

user@Ubuntu:~/hello$ **git log**

commit c37f6906ad6fc7e8f85456e271a3365ab32a35cb (HEAD -> master)

Author: user <“alm15@mail.ru”>

Date: Wed Oct 16 05:45:44 2024 +0000

first version

Узнать кто вносил изменения по времени

user@Ubuntu:~/hello$ **git log -p**

commit e0bd46424a2ef0524ea92913f60c00f17e8a8038 (HEAD -> master)

Author: “Almaskhan <“alm15@mail.ru”>

Date: Wed Oct 16 05:48:49 2024 +0000

second version

diff --git a/app.py b/app.py

index c8a7a58..eed5906 100755

--- a/app.py

+++ b/app.py

@@ -4,7 +4,7 @@ import datetime

def do\_magic():

now = datetime.datetime.now()

- return "Hello! {0}".format(now)

+ return "Hello brother! {0}".format(now)

if \_\_name\_\_ == "\_\_main\_\_":

print(do\_magic())

commit c37f6906ad6fc7e8f85456e271a3365ab32a35cb

Author: user <“alm15@mail.ru”>

Date: Wed Oct 16 05:45:44 2024 +0000

**first version**

diff --git a/app.py b/app.py

new file mode 100755

index 0000000..c8a7a58

--- /dev/null

+++ b/app.py

@@ -0,0 +1,11 @@

+#!/usr/bin/env python3

+

+import datetime

+

+def do\_magic():

+ now = datetime.datetime.now()

Устанавливаем Apache2:

user@Ubuntu:~/hello$ **sudo apt install apache2**

Проверяем страницу по умолчанию

user@Ubuntu:~/hello$ **ls /var/www/html/**

**index.html**

Запускаем браузер:

user@Ubuntu:~/hello$ **curl** [**http://localhost**](http://localhost)

Делаем символьную ссылку (ярлык) с www на каталог пользователя:

user@Ubuntu:~/hello$ **cd /var/www/**

Добавляем пользователя www-data в группу нашего пользователя user, чтобы веб сервер мог читать домашний каталог пользователя /home/user/hello:

# **sudo usermod www-data -a -G user**

Если не добавить, то будет ошибка **<p>You don't have permission to access this resource.</p>**

Перезапускаем веб сервер чтобы он запустился с новыми правами:

user@Ubuntu:/var/www$ **systemctl restart apache2**

Переносим каталог веб сервера в каталог пользователя:

user@Ubuntu:/var/www$ **sudo mv html html2**

user@Ubuntu:/var/www$ **sudo ln -s ~/hello/ html**

user@Ubuntu:/var/www$ **ls -la**

total 12

drwxr-xr-x 3 root root 4096 Oct 16 05:59 .

drwxr-xr-x 14 root root 4096 Oct 16 05:53 ..

lrwxrwxrwx 1 root root 17 Oct 16 05:59 *html -> /home/user/hello/*

drwxr-xr-x 2 root root 4096 Oct 16 05:53 html2

Проверяем страницу:

user@Ubuntu:/var/www$ **curl** [**http://localhost**](http://localhost)

Включаем CGI для веб страниц:

user@Ubuntu:/var/www$ **sudo a2enmod cgi**

user@Ubuntu:/var/www$ **sudo vim /etc/apache2/sites-enabled/000-default.conf**

<Directory /var/www/html>

AllowOverride All

<Directory>

Перезапускаем веб сервер:

$ **systemctl restart apache2**

Заходим в каталог hello с приложением и создаем файл .htaccess

$ **cd ~/hello**

$ **vi .htaccess**

AddHandler cgi-script .py

Options +ExecCGI

DirectoryIndex app.py

Меняем наш app.py чтобы время отображалось в браузере, добавляем строчку:

$ **vi app.py**

#!/usr/bin/env python3

import datetime

def do\_magic():

now = datetime.datetime.now()

return "Hello! {0}".format(now)

if \_\_name\_\_ == "\_\_main\_\_":

**print("Content-type: text/html\n\n")**

print(do\_magic())

Если выходить ошибка при заходе на страницу <http://localhost>

user@Ubuntu:~$ **curl** [**http://localhost**](http://localhost)

И видим строчку:

Hello! 2024-10-16 12:35:56.184795

Что в консоли не видеть **Content-type: text/html** меняем **в app.py**

$ **vi app.py**

#!/usr/bin/env python3

import datetime

**import os**

def do\_magic():

now = datetime.datetime.now()

return "Hello! {0}".format(now)

if \_\_name\_\_ == "\_\_main\_\_":

print("Content-type: text/html\n\n")

**print(os.environ)**

print(do\_magic())

После заходим на страничку app.py

user@Ubuntu:~$ **curl** [**http://localhost**](http://localhost)

И видим строчку где переменные с настройками OS, здесь нам интересна переменная REQUEST\_URI которая нам приезжает если меня вызывает веб сервер Apache:

Content-type: text/html

environ({'SHELL': '/bin/bash', 'PWD': '/home/user/hello', 'LOGNAME': 'user', 'XDG\_SESSION\_TYPE': 'tty', 'MOTD\_SHOWN': 'pam', 'HOME': '/home/user', 'LANG': 'en\_US.UTF-8', 'LS\_COLORS':

Hello! 2024-10-16 12:35:56.184795

Поэтому мы можем сделать универсальный код, если вызывает нас Apache, то мы показываем строчку Content-type, если нет не выводим строчку.

$ **vi app.py**

#!/usr/bin/env python3

import datetime

import os

def do\_magic():

now = datetime.datetime.now()

return "Hello! {0}".format(now)

if \_\_name\_\_ == "\_\_main\_\_":

**if 'REQUEST\_URI' in os.environ:**

**print("Content-type: text/html\n\n")**

print(do\_magic())

Проверяем и не видим content-type:

user@Ubuntu:~/hello$ **./app.py**

Hello! 2024-10-16 12:53:34.196540

user@Ubuntu:~/hello$ **curl** [**http://localhost**](http://localhost)

Hello! 2024-10-16 12:54:18.343903

Теперь можем посмотреть статус git status и видим что модицифицирован app.py, и пора добавить .httaccess потому-что он часть когда:

$ **git status**

Добавляем файл htacces и app.py в репозиторий. И делаем пометку 'apache2 cgi version'

user@Ubuntu:~/hello$ **git add .htaccess**

user@Ubuntu:~/hello$ **git** **add app.py**

user@Ubuntu:~/hello$ **git commit -m 'apache2 cgi version'**

Теперь можно пометить приложение как некоторую версию, даем команду git tag v1.0. Последний наш комит теперь будет с ярлычком v1.0 (тэгом) и мы можем с помощью тэгов быстро возвращаться к разным версиям приложения.

user@Ubuntu:~/hello$ **git tag v1.0**

**С помощью команды tag мы добавили тэг к всем комитам**

user@Ubuntu:~/hello$ **git log**

**commit 8beb9764aea72f7b4c394e18ac2c5c7da6923958 (HEAD -> master, tag: v1.0)**

Author: “Almaskhan <alm15@mail.ru>

Date: Wed Oct 16 12:57:43 2024 +0000

**apache2 cgi version**

**commit f8db7338feaffa379940462c37a8ddcfe7f8ba9e**

Author: “Almaskhan <alm15@mail.ru>

Date: Wed Oct 16 12:42:32 2024 +0000

**cgi version**

**commit bfac3736772a71b6714247a9005eeda245aa8632**

Author: “Almaskhan <alm15@mail.ru>

Date: Wed Oct 16 12:14:08 2024 +0000

**first version’**

Например давайте вернемся к превому комиту, берем последние 12 цифр комита и вставляем в checkout:

user@Ubuntu:~/hello$ **git checkout bfac3736772a71b**

Проверяем наш **app.py**:

user@Ubuntu:~/hello$ **cat app.py**

#!/usr/bin/env python3

import datetime

def do\_magic():

now = datetime.datetime.now()

return "Hello! {0}".format(now)

if \_\_name\_\_ == "\_\_main\_\_":

print(do\_magic())

Также проверяем состояние файлов в каталоге:

user@Ubuntu:~/hello$ **ls**

app.py

И видим, что он откатился к первой версии которая была в начале. Git также привел содержимое каталога в первоначальное состояние. То есть теперь там нет файла .htaccess

Теперь можем вернуться к последней версии:

user@Ubuntu:~/hello$ **git checkout v1.0**

Проверяем наш app.py:

user@Ubuntu:~/hello$ **cat app.py**

#!/usr/bin/env python3

import datetime

import os

def do\_magic():

now = datetime.datetime.now()

return "Hello! {0}".format(now)

if \_\_name\_\_ == "\_\_main\_\_":

if 'REQUEST\_URI' in os.environ:

print("Content-type: text/html\n\n")

print(do\_magic())

**Гит позволяет вернутся к самой последней версии кода** через команду **checkout master**

(тэг нужен для определенной версии):

user@Ubuntu:~/hello$ **git checkout master**

**ЧТО МЫ УВИДЕЛИ:**

* **Git помогает двигаться по версиям кода и понимать, что происходит**
* **Некоторые коммиты мы можем помечать особо (тэги)**

С помощью команды git tag мы можем увидеть все тэги в репозитории приложения:

user@Ubuntu:~/hello$ **git tag**

v1.0

v1.5

v2.0

Можно сказать релиз инженеру чтобы он прогнал тесты с тэгом v1.0, а другие не трогай.

**А ДАЛЬШЕ?**

* **Ну, CGI не круто, каждый запрос свой процесс**
* **Для Python стильно модно и молодёжно использовать WSGI (whiskey)**
* **В качестве софтовой реализации м можем использовать uWSGI**

CGI это не круто, для Питон стильно модно молодежно использовать Whiskey, это сервер который умеет выполнять Питон файлы, умеет их выполнять масштабируясь в необходимое количество потоков процессов и т. д. И реализация есть в специальном демоне UWSKY (you whiskey – ё виски).

Сейчас будем имзенять репозиторий, чтобы не мешать коллегам мы можем сделать копию репозиотория со всем его кодом, в которую дальше будем складывать свои комиты. Когда будем довольны ее содержимым, будем считать разработку данной фичи законченной, вольем наши изменения в основной репозиторий.

С помощью команды делаем копию репозитория (branch – другая ветка с именем uwsgi), чтобы другие инженеры не видели нашу работу:

user@Ubuntu:~/hello$ **git checkout -b uwsgi**

Теперь в нем можно делать все что мы захотим. Это удобно для работы с коллегами параллельно не мешая друг другу. Интересно будет когда мы захоти когда ветки нужно слить вместе.

Для совместимости со стандартом Whiskey нужно ввести в приложение ввести следующею процедуру:

#!/usr/bin/env python3

import datetime

import os

def do\_magic():

now = datetime.datetime.now()

return "Hello! {0}".format(now)

**def application(env, start\_response):**

**start\_response('200 OK',[('Content-Type','text/html')])**

**return [do\_magic()]**

if \_\_name\_\_ == "\_\_main\_\_":

if 'REQUEST\_URI' in os.environ:

print("Content-type: text/html\n\n")

print(do\_magic())

Далее добавляем наш app.py в репозиторий:

user@Ubuntu:~/hello$ **git add app.py**

Далее коммитим и добавляем к коммиту uwsgi version (виски версия):

user@Ubuntu:~/hello$ **git commit -m 'uwsgi version'**

Теперь устанавливаем то, что нужно для работы этой версии, виски — это особый сервер:

user@Ubuntu:~/hello$ **sudo apt install uwsgi-plugin-python3**

Это установит сам Виски демон и нужный к нему плагин, который позволит запускать скрипты в нем.

Чтобы убедится что демон uwsgi установился, попытаемся его запустить:

user@Ubuntu:~/hello$ **uwsgi**

В ответ видим вывод.

user@Ubuntu:~/hello$ **uwsgi --plugin python3 --http-socket :9090 --wsgi-file app.py**

**Запускаем браузер и проверяем:**

user@Ubuntu:~/hello$ **сurl** [**http://localhost:9090**](http://localhost:9090)

И видим пустую страницу, чтобы работало нужно изменить строчку

#!/usr/bin/env python3

import datetime

import os

def do\_magic():

now = datetime.datetime.now()

return "Hello! {0}".format(now)

def application(env, start\_response):

start\_response('200 OK',[('Content-Type','text/html')])

**return [do\_magic().encode()]**

if \_\_name\_\_ == "\_\_main\_\_":

if 'REQUEST\_URI' in os.environ:

print("Content-type: text/html\n\n")

print(do\_magic())

**Запускаем браузер и проверяем:**

user@Ubuntu:~/hello$ **сurl** [**http://localhost:9090**](http://localhost:9090)

Чтобы не запускать демон с длинной строкой в консоли, мы можем создать ini файл dev.ini со след. содержимым:

[uwsgi]

plugin=python3

http-socket=:9090

wsgi-file=app.py

И теперь можно запускать стильно модно молодежно:

user@Ubuntu:~/hello$ **uwsgi dev.ini**

**Проверяем статус нашей рабочей директории:**

user@Ubuntu:~/hello$ **git status**

On branch uwsgi

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

**modified: app.py**

Untracked files:

(use "git add <file>..." to include in what will be committed)

**dev.ini**

no changes added to commit (use "git add" and/or "git commit -a")

Видим, что изменились файлы app.py и появился dev.ini.

user@Ubuntu:~/hello$ **git add app.py**

user@Ubuntu:~/hello$ **git add dev.ini**

Добавляем их в репозиторий с обязательным комитом:

user@Ubuntu:~/hello$ **git commit -m 'uwsgi version'**

Проверяем ветки и видим что текущая ветка uwsgi:

user@Ubuntu:~/hello$ **git branch**

master

\* uwsgi

Теперь сливаем нашу ветку uwsgi с веткой мастер, возвращаем файлы с мастер ветки:

user@Ubuntu:~/hello$ **git checkout master**

Switched to branch 'master'

user@Ubuntu:~/hello$ **git merge uwsgi**

Updating 8beb976..4b4a648

Fast-forward

app.py | 5 +++++

dev.ini | 5 +++++

2 files changed, 10 insertions(+)

create mode 100644 dev.ini

Запускаем git log для просмотра истории изменений и видим что наши изменения сели в мастер ветку:

user@Ubuntu:~/hello$ **git log**

commit 4b4a64874a77db81e7a294fc7464dd5dad223b78 (**HEAD -> master, uwsgi)**

Author: “Almaskhan <alm15@mail.ru>

Date: Fri Oct 18 09:08:31 2024 +0000

**uwsgi version**

commit 6021330ba670e0f79d4419d924c45377de270231

Author: “Almaskhan <alm15@mail.ru>

Date: Fri Oct 18 08:18:02 2024 +0000

**uwsgi version**

commit 8beb9764aea72f7b4c394e18ac2c5c7da6923958 (tag: v1.0)

Author: “Almaskhan <alm15@mail.ru>

Date: Wed Oct 16 12:57:43 2024 +0000

**apache2 cgi version**

commit f8db7338feaffa379940462c37a8ddcfe7f8ba9e

Author: “Almaskhan <alm15@mail.ru>

Date: Wed Oct 16 12:42:32 2024 +0000

**cgi version**

commit bfac3736772a71b6714247a9005eeda245aa8632

Author: “Almaskhan <alm15@mail.ru>

Date: Wed Oct 16 12:14:08 2024 +0000

**first version’**

Запускаем git log -p для просмотра детальной истории:

user@Ubuntu:~/hello$ **git log -p**

**А ДАЛЬШЕ?**

* **Мы пока не закончили, у нас используется встроенный сервер**
* **А вообще, хорошо бы использовать nginx, например**
* **И шеф на переговорах, поэтому в воздухе пахнет скорым деплоем**

**Выключаем Apache2 и установим nginx:**

user@Ubuntu:~/hello$ **sudo systemctl disable apache2**

user@Ubuntu:~/hello$ **sudo systemctl stop apache2**

user@Ubuntu:~/hello$ **sudo apt install nginx -y**

Мы сейчас создадим простенький конфигурационный файл, который позволит nginx принимать входящие соединения, пробрасывать их на uwhiskey

user@Ubuntu:~/hello$ **cd /etc/nginx/sites-enabled**

user@Ubuntu:~/hello$ **ls -la**

Удаляем файл конфигураций по умолчанию:

user@Ubuntu:/etc/nginx/sites-enabled$ **sudo** **rm -rf default**

user@Ubuntu:/etc/nginx/sites-enabled$ **sudo vi hello.conf**

server {

listen 80;

root /var/www/html;

location / {

include /etc/nginx/uwsgi\_params;

uwsgi\_pass 127.0.0.1:9000;

uwsgi\_param Host $host;

uwsgi\_param X-Real-IP $remote\_addr;

uwsgi\_param X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

uwsgi\_param X-Forwarded-Proto $http\_x\_forwarded\_proto;

}

}

Проверяем конфигурационный файл на ошибки:

user@Ubuntu:/etc/nginx/sites-enabled$ **sudo nginx -t**

nginx: the configuration file /etc/nginx/nginx.conf syntax **is ok**

nginx: configuration file /etc/nginx/nginx.conf test is successful

По конфигу: uwsgi\_pass 127.0.0.1:9000; - отвечает за проброс на демон uwsgi по порту 9000 (ю виски)

Хранить hello.conf нужно в папке sites-available (в нем хранятся все конфиги). А в папке sites-enabled, у нас хранятся только рабочие конфиги. Поэтому перемещаем файл:

user@Ubuntu:/etc/nginx/sites-enabled$ **sudo mv hello.conf ../sites-available/**

И создаем ярлык ссылку с sites-enabled\hello.conf на sites-available\hello.conf (оригинал)

user@Ubuntu:/etc/nginx/sites-enabled$ **sudo ln -s ../sites-available/hello.conf .**

Проверяем:

user@Ubuntu:/etc/nginx/sites-enabled$ **pwd**

/etc/nginx/sites-enabled

user@Ubuntu:/etc/nginx/sites-enabled$ **ls -lha**

lrwxrwxrwx 1 root root 29 Oct 18 15:36 hello.conf -> ../sites-available/hello.conf

Перезапускаем nginx:

user@Ubuntu:/etc/nginx/sites-enabled$ **sudo systemctl restart nginx**

Заходим на сайт для проверки:

user@Ubuntu:/etc/nginx/sites-enabled$ **curl** [**http://localhost**](http://localhost)

<html>

<head><title>502 Bad Gateway</title></head>

<body>

<center><h1>502 Bad Gateway</h1></center>

<hr><center>nginx/1.18.0 (Ubuntu)</center>

</body>

</html>

И видим, что у нас сайт не открывается – ошибка 502 Bad Gateway

Это потому-что демон uwsgi (ю виски) не запущен.

Переходим в директорию с нашим кодом ~/hello

user@Ubuntu:/etc/nginx/sites-enabled$ **cd ~/hello**

Копируем dev в prod:

user@Ubuntu:~/hello$ **cp dev.ini prod.ini**

И изменяем prod файл:

user@Ubuntu:~/hello$ **vi prod.ini**

[uwsgi]

plugin=python3

socket=127.0.0.1:9000

wsgi-file=app.py

Здесь мы поменяли строку и добавили сокет (ip+порт)

Запускаем ювиски:

user@Ubuntu:~/hello$ **uwsgi prod.ini**

Проверяем наш сайт на доступность:

user@Ubuntu:/var/www$ **curl** [**http://localhost**](http://localhost)

**или**

user@Ubuntu:/var/www$ **curl http://10.0.10.113**

И получаем строку нашего сайта

Hello! 2024-10-18 15:51:59.81971

**А ДАЛЬШЕ?**

* **А тут шеф прибегает и говорит, что пора бы деплой делать, Клиент есть**
* **Нам предстоит писать не просто код, а инфраструктурный код**
  + **То есть, тот код который умеет выкатывать наше приложение**
  + **Не руками же это делать**
* **Пока мы решаем хранить инфраструктурный код вместе с кодом приложения**
  + **Это не всегда так, есть свои минусы и плюсы**

Идем в hello – cd ~/hello и создаем там каталоги:

user@Ubuntu:~/hello$ **sudo mkdir -p deploy/{apache2,uwsgi,nginx,systemd}**

Проверяем что каталоги создались:

user@Ubuntu:~/hello$ ls **-l deploy/**

total 16

drwxr-xr-x 2 root root 4096 Oct 18 15:55 apache2

drwxr-xr-x 2 root root 4096 Oct 18 15:55 nginx

drwxr-xr-x 2 root root 4096 Oct 18 15:55 systemd

drwxr-xr-x 2 root root 4096 Oct 18 15:55 uwsgi

Проверяем что файлы в каталоге hello

user@Ubuntu:~/hello**$ ls -lha**

total 36K

drwxrwxr-x 4 user user 4.0K Oct 18 15:55 .

drwxr-x--- 6 user user 4.0K Oct 18 15:51 ..

-rwxrwxr-x 1 user user 400 Oct 18 13:46 app.py

drwxr-xr-x 6 root root 4.0K Oct 18 15:55 deploy

-rw-rw-r-- 1 user user 59 Oct 18 13:46 dev.ini

drwxrwxr-x 8 user user 4.0K Oct 18 13:46 .git

-rw-rw-r-- 1 user user 66 Oct 18 13:46 .htaccess

-rw-r--r-- 1 root root 612 Oct 18 15:21 **index.nginx-debian.html**

-rw-rw-r-- 1 user user 63 Oct 18 15:48 prod.ini:

И видим файлик index.nginx-debian.html который появился при установке веб сервера nginx (это тестовая страница для проверки веб сервера). Его мы удаляем:

user@Ubuntu:~/hello$ **sudo rm index.nginx-debian.html**

Файлы в каталоге hello перекладываем в подпапки Deploy

-rwxrwxr-x 1 user user 400 Oct 18 13:46 app.py

drwxr-xr-x 6 root root 4.0K Oct 18 15:55 **deploy**

-rw-rw-r-- 1 user user 59 Oct 18 13:46 dev.ini -> перекладываем в каталог deploy/uwsgi

drwxrwxr-x 8 user user 4.0K Oct 18 13:46 .git

-rw-rw-r-- 1 user user 66 Oct 18 13:46 .htaccess -> перекладываем в каталог deploy/apache

-rw-rw-r-- 1 user user 63 Oct 18 15:48 prod.ini -> перекладываем в каталог deploy/uwsgi

Перемещаем файлы:

user@Ubuntu:~/hello$ **sudo mv .htaccess deploy/apache2/**

user@Ubuntu:~/hello$ **sudo mv \*.ini deploy/uwsgi/**

И файл hello.conf в каталоге /etc/nginx/sites-available/hello.conf в папку deploy/nginx

user@Ubuntu:~/hello$ **sudo cp /etc/nginx/sites-available/hello.conf ~/hello/deploy/nginx/**

И теперь директория стала красивой:

user@Ubuntu:~/hello$ **ls -lha**

total 20K

drwxrwxr-x 4 user user 4.0K Oct 18 16:07 .

drwxr-x--- 6 user user 4.0K Oct 18 15:57 ..

-rwxrwxr-x 1 user user 400 Oct 18 13:46 app.py

drwxr-xr-x 6 root root 4.0K Oct 18 15:55 deploy

drwxrwxr-x 8 user user 4.0K Oct 18 13:46 .git

Запускаем гит на проверку изменений каталога:

user@Ubuntu:~/hello$ **git status**

Видим, что много измененийб, добавляем каталог deploy и делаем коммит:

user@Ubuntu:~/hello$ **git add deploy**

user@Ubuntu:~/hello$ **git commit -am ‘prepare to deploy’**

Теперь выкладываем этот код на аккаунт на Гитхабе. То есть на внешний сервер.

По умолчанию связь с удаленным репозиторием по http/ssh - логин и паролю, поэтому нужно настраивать доступ по ключам. Если вдруг мы зашли по http/ssh логину, то нужно удалить эту связь и сделать вход по ключу.

user@Ubuntu:~/hello$ **git remote remove origin**

Настраиваем связь нашего локального репозитория с публичным на github по ssh ключам.

Чтобы связать локальный репозиторий Git с удалённым на GitHub с помощью SSH, выполните следующие шаги:

### 1. ****Создание SSH-ключа (если ещё не создан)****

Если у вас еще нет SSH-ключа, выполните команду для его создания:

bash

Копировать код

ssh-keygen -t rsa -b 4096 -C "your\_email@example.com"

* Выберите местоположение для хранения ключа (по умолчанию — ~/.ssh/id\_rsa).
* Установите парольную фразу, если хотите (можно пропустить).

### 2. ****Добавьте SSH-ключ в ssh-agent****

Активируйте ssh-agent и добавьте ваш приватный ключ:

bash

Копировать код

eval "$(ssh-agent -s)"

ssh-add ~/.ssh/id\_rsa

### 3. ****Добавьте SSH-ключ на GitHub****

* Скопируйте ваш публичный ключ в буфер обмена:

bash

Копировать код

cat ~/.ssh/id\_rsa.pub

* Перейдите на [страницу добавления SSH-ключа в GitHub](https://github.com/settings/keys).
* Нажмите на кнопку **New SSH key**, вставьте ключ в соответствующее поле и нажмите **Add SSH key**.

### 4. ****Проверьте подключение к GitHub через SSH****

Убедитесь, что SSH-ключ работает:

bash

Копировать код

ssh -T git@github.com

Если всё настроено правильно, вы получите сообщение вроде:

vbnet

Копировать код

Hi username! You've successfully authenticated, but GitHub does not provide shell access.

### 5. ****Свяжите локальный репозиторий с удалённым****

Теперь нужно добавить удалённый репозиторий на GitHub в качестве удалённого для вашего локального репозитория.

* Перейдите в локальный репозиторий:

bash

Копировать код

cd /путь/к/вашему/репозиторию

* Свяжите его с удалённым репозиторием на GitHub с помощью SSH:

bash

Копировать код

git remote add origin git@github.com:username/repo.git

Где username — ваше имя пользователя на GitHub, а repo — имя вашего репозитория.

### 6. ****Проверьте связь с удалённым репозиторием****

Убедитесь, что удалённый репозиторий настроен:

bash

Копировать код

git remote -v

Вы должны увидеть что-то вроде:

scss

Копировать код

origin git@github.com:username/repo.git (fetch)

origin git@github.com:username/repo.git (push)

### 7. ****Отправка изменений****

Теперь можно отправить изменения в удалённый репозиторий:

bash

Копировать код

git push -u origin master

Если у вас не настроена ветка master, замените её на нужную вам ветку.

После этого локальный репозиторий будет связан с удалённым на GitHub через SSH

user@Ubuntu:~/hello$ **ssh -T git@github.com**

Hi almsys! You've successfully authenticated, but GitHub does not provide shell access.

user@Ubuntu:~/hello$ **git remote add origin git@github.com:almsys/hello.git**

user@Ubuntu:~/hello$ **git push origin master**

Теперь наш каталог ~/hello/\* синхронизировался с публичным github/almsys/hello.

Но теперь наша история изменений не скопировалась, поэтому мы можем сами перенести тэг v1.0

user@Ubuntu:~/hello$ **git push origin v1.0**

Наша демон ювиски (uwsgi) запускается в интерактивном режиме, что очень не удобно. Поэтому давайте создадим system unit для запуска демона

user@Ubuntu:~/hello$ **sudo vi /etc/systemd/system/hello.service**

[Unit]

Description=Hello app

Requires=network.target

After=network.target

[Service]

TimeoutStartSec=0

RestartSec=10

Restart=alwways

WorkingDirectory=/opt/hello

KillSignal=SIGQUIT

Type=notify

NotifyAccess=all

ExecStart=/usr/bin/uwsgi deploy/uwsgi/prod.ini

[Install]

WantedBy=multi-user.target

Теперь заходим в /opt и клонируем репозиторий

user@Ubuntu:~/hello$ **cd /opt**

user@Ubuntu:/opt$ **git clone https://github.com/almsys/hello.git**

user@Ubuntu:/opt$ **sudo systemctl start hello**

Теперь у нас работает демон hello

user@Ubuntu:/opt$ **sudo systemctl status hello**

Заходим на наш сайт:

user@Ubuntu:/opt$ **curl http://localhost**

Hello! 2024-10-19 09:43:31.168593user@Ubuntu:/opt$

И видим, что он работает.

user@Ubuntu:/opt$ **cd ~/hello**

user@Ubuntu:~/hello$ **cp /etc/systemd/system/hello.service deploy/systemd/**

У нас готовы все файлы для деплоя на любой сервер в бой.

Нужно чтобы был установлен:

1. Питон
2. Nginx
3. Uwsgi
4. И чтобы были выложены файлики в репозитории

user@Ubuntu:~/hello$ **git add deploy/\***

user@Ubuntu:~/hello$ **git commit -m 'finished deploy'**

user@Ubuntu:~/hello$ **git tag v2.0**

user@Ubuntu:~/hello$ **git push origin master**

user@Ubuntu:~/hello$ **git push origin v2.0**

**А ДАЛЬШЕ?**

* **Мы приготовили нужные артефакты к деплою**
* **Поскольку инфра-код у нас часть репозитория? То изменения в инфра-коде тоже двигают версию**

Теперь установим LXD – линуксовые контейнеры

У нас есть версия кода v2.0 для релиза, теперь нам надо автоматизировать выход релиза (деплой).

Это можно сделать при помощи контейнеров. Мы сделаем контейнер, который будем настраивать с помощью ansible, и я покажу вам штуку как управление конфигурацией.

Первоначальная настройка LXD, нам будут задавать вопросы, на все отвечаем ENTER (по умолчанию) и только меняем настройку для хранения данных как каталог **dir**:

user@Ubuntu:~$ **sudo lxd init**

Would you like to use LXD clustering? (yes/no) [default=**no**]:

Do you want to configure a new storage pool? (yes/no) [default=**yes**]:

Name of the new storage pool [default=**default**]:

Name of the storage backend to use (zfs, btrfs, ceph, cephobject, **dir,** lvm) [default=zfs]: **dir**

Would you like to connect to a MAAS server? (yes/no) [default**=no**]:

Would you like to create a new local network bridge? (yes/no) [default**=yes**]:

What should the new bridge be called? [default**=lxdbr0**]:

What IPv4 address should be used? (CIDR subnet notation, “auto” or “none”) [default**=auto**]:

What IPv6 address should be used? (CIDR subnet notation, “auto” or “none”) [default**=auto**]:

Would you like the LXD server to be available over the network? (yes/no) [default**=no**]:

Would you like stale cached images to be updated automatically? (yes/no) [default**=yes**]:

Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default**=no**]:

Теперь есть возможность создавать контейнеры, но, если будем создавать он будет пустой.

Чтобы этого не было нам нужно туда положить публичный ключ.

Делаем выход (logout) из системы и вход (login), потому-что нашего пользователя user добавили в группу lxd, это работает при пере-заходе:

user@Ubuntu:~$ **logout**

Дальше делаем редактирование дефолтной конфигурации:

user@Ubuntu:~/hello$ **lxc profile edit default**

### This is a YAML representation of the profile.

### Any line starting with a '# will be ignored.

###

### A profile consists of a set of configuration items followed by a set of

### devices.

###

### An example would look like:

### name: onenic

### config:

### raw.lxc: lxc.aa\_profile=unconfined

### devices:

### eth0:

### nictype: bridged

### parent: lxdbr0

### type: nic

###

### Note that the name is shown but cannot be changed

config: {}

description: Default LXD profile

devices:

eth0:

name: eth0

network: lxdbr0

type: nic

root:

path: /

pool: default

type: disk

name: default

used\_by: []

Меняем дефолтный конфиг и добавляем туда наш публичный который можно получить, введя команду - **cat ~/.ssh/id\_rsa.pub**

Это нужно чтобы новые контейнеры создавались сразу с ключом доступа в каталоге ~/.ssh

**config:**

**cloud-init.user-data: |**

**#cloud-config**

**packages:**

**- openssh-server**

**- vim**

**ssh\_pwauth: false**

**ssh\_authorized\_keys:**

**- "ssh-rsa KEY HERE”**

description: Default LXD profile

devices:

eth0:

name: eth0

network: lxdbr0

type: nic

root:

path: /

pool: default

type: disk

name: default

used\_by: []

Повторная команда edit проверяем конфиг, если что сразу скажет что не так:

user@Ubuntu:~/hello$ **lxc profile edit default**

Бывает что нужно создать другой конфиг для создания конте йнеров, например для продуктовой и тестовой среды:

user@Ubuntu:~/hello$ **profile create prod01**

user@Ubuntu:~/hello$ **profile create test01**

Далее можно создать файл Yaml с настройками для продуктовой среды, например такой:

user@Ubuntu:~/hello$ **vi** **config.yml**

config:

cloud-init.user-data: |

#cloud-config

packages:

- openssh-server

- vim

ssh\_pwauth: false

ssh\_authorized\_keys:

- "ssh-rsa  alm15@mail.ru"

description: Default LXD profile

devices:

eth0:

name: eth0

network: lxdbr0

type: nic

root:

path: /

pool: default

type: disk

name: default

used\_by: []

Далее можно загрузить туда созданный конфиг в Prod

user@Ubuntu:~/hello$ **lxc profile edit prod < config.yml**

И после можно создавать машину используя новый конфиг:

user@Ubuntu:~/hello$ **lxc launch ubuntu: u4 --profile prod**

Здесь мы создали контейнер u4 с настройками для prod среды. Пользователь внутри контейнера **по умолчанию ubuntu**, но можно создать нового, например пользователя **ansible**:

config:

cloud-init.user-data: |

#cloud-config

packages:

- openssh-server

- vim

ssh\_pwauth: false

users:

- name: **ansible**

gecos: Ansible User

groups: users,admin,wheel

sudo: ALL=(ALL) NOPASSWD:ALL

shell: /bin/bash

ssh\_authorized\_keys:

- "ssh-rsa == alm15@mail.ru"

description: Default LXD profile

Добавление ключа в конфиг позволит быстренько авторизовываться, минуя логина и пароля.

Еще один способ сохранить ключ в контейнере:

**user@Ubuntu:~/hello/ansible$** lxc file push ~/.ssh/id\_rsa.pub main-caiman/home/ubuntu/.ssh/authorized\_keys

**user@Ubuntu:~/hello/ansible$** ssh -T ubuntu@10.169.189.95

Теперь создаем контейнер с новой Ubuntu:

user@Ubuntu:~/hello$ **lxc launch ubuntu:**

Проверяем входим по ключу который записывали в yaml файл:

user@Ubuntu:~/hello$ **ssh -T** [**ubuntu@10.169.189.127**](mailto:ubuntu@10.169.189.127)

**Если не заходит, значить наш публичный ключ cat ~/.ssh/id\_rsa.pub не был добавлен в файлик домашнего каталога** **/home/ubuntu/.ssh/authorized\_keys**

user@Ubuntu:~/hello$ **lxc exec main-caiman /bin/bash**

root@main-caiman:~# **su – ubuntu**

ubuntu@main-caiman:~$ **vi ~/.ssh/authorized\_keys**

И теперь устанавливаем Ansible для управления конфигураецией. Для этого устанавливаемsoftware-properties-common.

Для начала обновляем ОС Ubuntu:

user@Ubuntu:~/hello$ **sudo apt update -y**

И ставим Personal Package Archive коротко PPA (репозиторий по типу Epel в CentOS):

user@Ubuntu:~/hello$ **sudo apt install -y software-properties-common**

Это достаточно общий способ установки пакетов в Ubuntu. Подключаю Ansible PPA:

user@Ubuntu:~/hello$ **sudo apt-add-repository --yes --update ppa:ansible/ansible**

И теперь ставим Ansible:

user@Ubuntu:~/hello$ **sudo apt install ansible -y**

В чем важность системы управления конфигурацией для Devops – потому Devops это про ускорение проекта, а ни что так не ускоряет как автоматизации, нам нужно автоматизировать выкладки нашего приложения.

Проверяем наш контейнер LXC, смотри что он есть:

user@Ubuntu:~/hello$ **lxc list**

+-------------+---------+-----------------------+-----------------------------------------------+-----------+-----------+

| NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS |

+-------------+---------+-----------------------+-----------------------------------------------+-----------+-----------+

| **main-caiman |** RUNNING | 10.169.189.127 (eth0) | fd42:2f9c:e587:1a77:216:3eff:fe16:ea1b (eth0) | CONTAINER | 0 |

+-------------+---------+-----------------------+-----------------------------------------------+-----------+-----------+

Видим виртуалку с именем **main-caiman** с ip адресом 10.169.189.127

Контейнер это виртуалка, только у нее нет своего ядра, ядро оно берет основной системы. Из-за этого контейнер очень быстрый и он может легко пересоздаваться и быстро перезапускаться.

Контейнеры — это первый шажок к докеру.

Чтобы посмотреть что это такое контейнер, можно зайти в ее командную строку через команду:

user@Ubuntu:~/hello/ansible$ **lxc exec main-caiman /bin/bash**

Можно проверить версию операционной системы контейнера:

root@main-caiman**:~**# **lsb\_release -a**

No LSB modules are available.

Distributor ID: Ubuntu

Description: Ubuntu 24.04.1 LTS

Release: 24.04

Codename: noble

Проверяем есть ли ядро, обычно у контейнеров нету своего ядра, потому-что они используют ядро хост системы:

root@main-caiman:~# **ls /boot/**

И видим, что каталог пустой. Обычно там лежат образы ядра.

Что мы дальше делаем, мы идем в свою директорию с проектом. И в нем создаем директорию Ansible, потому-что там хочу держать скрипты, которыми будем раскатывать свое приложение, я хочу это делать автоматически.

user@Ubuntu:~/hello$ **cd ~/hello**

user@Ubuntu:~/hello$ **mkdir ansible**

user@Ubuntu:~/hello**$ cd ansible**

Создаем файл anisble.cfg где описываем где брать хосты для установки:

user@Ubuntu:~/hello/ansible$ **nano ansible.cfg**

**[defaults]**

**inventory = hosts**

**host\_key\_checking = False**

**interpreter\_python=auto\_silent**

Здесь мы указали в инвентори, что хосты для управления нужно брать из файла hosts.

Узнаем IP нашего контейнера через lxc list:

user@Ubuntu:~/hello/ansible$ **lxc list**

+-------------+---------+-----------------------+-----------------------------------------------+-----------+-----------+

| NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS |

+-------------+---------+-----------------------+-----------------------------------------------+-----------+-----------+

| main-caiman | RUNNING | **10.169.189.127 (**eth0) | fd42:2f9c:e587:1a77:216:3eff:fe16:ea1b (eth0) | CONTAINER | 0 |

+-------------+---------+-----------------------+-----------------------------------------------+-----------+-----------+

Теперь создаем файл hosts:

user@Ubuntu:~/hello/ansible$ **nano hosts**

test ansible\_host=**10.169.189.127** ansible\_user=ubuntu

Здесь мы указали хост test с IP с lxc list.

Теперь делаем тестовый пинг на наш контейнер:

user@Ubuntu:~/hello/ansible$ **ansible -m ping test**

test | SUCCESS => {

"ansible\_facts": {

"discovered\_interpreter\_python": "/usr/bin/python3.12"

},

"changed": false,

"ping": "pong"

Видим что все работает.

Проверка соединения по SSH, узнаем список машин и подключаемся к нашему контейнеру:

user@Ubuntu:~$ **lxc list**

+-------------+---------+-----------------------+-----------------------------------------------+-----------+-----------+

| NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS |

+-------------+---------+-----------------------+-----------------------------------------------+-----------+-----------+

| **main-caiman** | RUNNING | **10.169.189.127** (eth0) | fd42:2f9c:e587:1a77:216:3eff:fe16:ea1b (eth0) | CONTAINER | 0 |

+-------------+---------+-----------------------+-----------------------------------------------+-----------+-----------+

user@Ubuntu:~$ **cd ~/hello/ansible/**

user@Ubuntu:~$ **nano deploy.yml**

---

- name: deploy hello app

gather\_facts: false

hosts: test

become: true

vars:

repo: https://github.com/maniaque/hello

repo\_dir: '/opt/hello'

packages:

- nginx

- python3

- uwsgi-plugin-python3

tasks:

- name: update apt cache

apt:

update\_cache: yes

- name: install packages

package:

name: "{{ item }}"

state: latest

with\_items: "{{ packages }}"

- name: checkout repo

git:

repo: "{{ repo }}"

dest: "{{ repo\_dir }}"

version: v2.0

- name: copy systemd config

copy:

remote\_src: yes

src: "{{ repo\_dir }}/deploy/systemd/hello.service"

dest: "/etc/systemd/system/hello.service"

- name: enable and start service

systemd:

name: hello

daemon\_reload: yes

enabled: yes

state: started

- name: copy nginx config

copy:

remote\_src: yes

src: "{{ repo\_dir }}/deploy/nginx/hello.conf"

dest: "/etc/nginx/sites-available/hello.conf"

- name: disable default nginx config

file:

state: absent

path: /etc/nginx/sites-enabled/default

notify: restart nginx

- name: enable our nginx config

file:

src: /etc/nginx/sites-available/hello.conf

dest: /etc/nginx/sites-enabled/hello.conf

state: link

notify: restart nginx

handlers:

- name: restart nginx

systemd:

name: nginx

state: restarted

Обязательно проверяем наш плейбук чтобы в нем не было ошибок:

user@Ubuntu:~/hello/ansible$ **ansible-playbook --syntax-check deploy.yml**

Теперь запускаем плейбук чтобы накатить настройки на наш контейнер:

user@Ubuntu:~/hello/ansible$ **ansible-playbook deploy.yml**

Далее проверяем что наше приложение hello app работает:

user@Ubuntu:~/hello/ansible$ **wget -O- http://10.169.189.218 | less**

--2024-10-27 06:50:25-- http://10.169.189.218/

Connecting to 10.169.189.218:80... connected.

HTTP request sent, awaiting response... 200 OK

Length: unspecified [text/html]

Saving to: ‘STDOUT’

- [ <=> ] 33 --.-KB/s in 0s

2024-10-27 06:50:25 (427 KB/s) - written to stdout [33]

Hello! 2024-10-27 06:50:25.882634

Тоже самое через curl:

user@Ubuntu:~/hello/ansible$ **curl http://10.169.189.218 | less**

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 33 0 33 0 0 4064 0 --:--:-- --:--:-- --:--:-- 4714

Hello! 2024-10-27 06:49:51.471250

Наш скрипт, деплоя раскатал все что нам надо, это очень клево!

Это заслуживает отдельного комита, мы идем в директорию:

user@Ubuntu:~/hello/ansible$ **cd ~/hello**

Говорю:

user@Ubuntu:~/hello$ **git status**

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

**ansible/**

nothing added to commit but untracked files present (use "git add" to track)

Говорю, что добавька мне директорию ansible

user@Ubuntu:~/hello$ **git add ansible/**

user@Ubuntu:~/hello$ **git commit -m ‘ansible deploy’**

Говорю, выложи мне это на github, чтобы было стильно модно и молодежно:

user@Ubuntu:~/hello$ **git push origin master**

**ЧТО МЫ УВИДЕЛИ?**

* **Мы использовали Ansible для автоматизации деплоя**
* **Поскольку нам нужен был максимально упрощённый способ, мы вязли контейнеры**
  + **У нас простой случай, мы не отрабатывали пересоздания контейнера**
  + **Для этого хорошо подходят снапшоты, мы не будем это показывать**

**А что дальше?**

* **Docker, конечно же**
  + **Overlayfs, слои позволяют строить образы поверх образов**
  + **Dockerfile**

Устанавливаем Docker, скачиваем установочный скрипт через консольный веб браузер curl:

user@Ubuntu:~$ **curl -fsSL https://get.docker.com -o get-docker.sh**

Запускаем скрипт:

user@Ubuntu:~$ **sudo sh get-docker.sh**

**Почему Docker?**

* **Простая сборка образов – контейнеров**
* **Можно строить контейнеры поверх других**
* **Dockerfile позволяет экономить на администрировании**
* **Docker Hub позволяет быстро делиться образами**
  + **При этом можно использовать свой, локальный**
  + **А можно использовать глобальный**

Docker это клево: Во первых Docker позволяет нам упаковывать софт в готовые аккуратные образы, мы подготавливаем образ контейнера который потом едет в development, production, к черту на кулички. Потому-что у нас есть Git – чтобы быть источником единой правды и Docker – источник package’а.

Докер позволяет упаковывать софт контейнеры – это селф контейнер штука, запусти ее на любой машине и будет хорошо. Контейнеры можно строить поверх других, то есть взяв за основу один контейнер, можно добавить свои изменения и построить новую версию его. Докер файл описывает шаги которые нужны для того чтобы запустить ваше приложение, разработчик пишет докер файл и позволяет сэкономить на администрировании. А докер хаб позволяет легко делится образами контейнеров, причем докер хаб можно использовать как локальный внутри организации, так и глобальный.

Чтобы постоянно не использовать sudo docker, нам нужно добавить своего пользователя в группу docker’a

user@Ubuntu:~$ **sudo usermod -aG docker user**

Теперь нужно перелогинтся чтобы настройки вступили в силу:

user@Ubuntu:~$ **logout**

Теперь нужно создать файл Dockerfile:

user@Ubuntu:~$ **cd hello/**

В нем пишем, что нам нужен образ tiangolo c uwsgi, nginx и python.

user@Ubuntu:~/hello$ **nano Dockerfile**

FROM tiangolo/uwsgi-nginx:python3.7

COPY ./app.py /app/app.py

COPY ./deploy/uwsgi/prod.ini /app/uwsgi.ini

WORKDIR /app

По документации к образу tianholo, приложение питон должно быть в директории app, а настройки uwsgi должны хранится там же. Рабочая директория /app

Теперь сравним настройку через Ansible и использования готового контейнера tiangolo. На Ansible мы писали длинный файл deploy.yml, а через готовый контейнер нам нужно всего 4 строчки для деплоя приложения.

Теперь собираем образ из нашего докер файла в текущей директории. Для начала докер скачиваем образ tiangolo и добавляет туда наши настройки из Dockerfile:

user@Ubuntu:~/hello$ **docker build -t almsys/hello:1.0 .**

Контейнеры не безопасны, чтобы убедится, что они безопасны, существует множество сканеров чтобы убедиться, что там все хорошо. Но при использовании контейнера он

**Почему не любят Docker?**

* **Не всегда лучшие практики администрирования**
* **Не всегда Dockerfile для разработки подходи для нормального прода**
  + **Экономия на администрировании иногда выходит боком**

Но чаще всего в докере все нормально…

Следующий шаг, который нам нужно сделать, это поставить Docker Compose.

user@Ubuntu:~/hello$ **VERSION=v2.30.0; sudo curl -Lo /usr/local/bin/docker-compose https://github.com/docker/compose/releases/download/$VERSION/docker-compose-`uname -s`-`uname -m` && sudo chmod +x /usr/local/bin/docker-compose**

Теперь делаем комит, добавляем ново созданный файл Dockerfile:

user@Ubuntu:~/hello**$ git status**

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

Dockerfile

nothing added to commit but untracked files present (use "git add" to track)

user@Ubuntu:~/hello$ **git add Dockerfile**

user@Ubuntu:~/hello$ **git commit -m 'Dockerfile'**

[master 6842556] Dockerfile

1 file changed, 6 insertions(+)

create mode 100644 Dockerfile

user@Ubuntu:~/hello$ **git tag v3.0**

user@Ubuntu:~/hello$ **git push origin v3.0**

Enumerating objects: 4, done.

Counting objects: 100% (4/4), done.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 433 bytes | 433.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

To github.com:almsys/hello.git

\* [new tag] v3.0 -> v3.0

Что дальше, собственно, у нас построился контейнер, теперь мы можем запустить контейнер и это сложно:

user@Ubuntu:~/hello**$ docker container run --publish 80:80 --detach almsys/hello:1.0**

a95e67f42130091e29ff5e223b055bc3c8bdde87c2b0346537f9266c8d7ddadb

Что что произогшло – он запустился. Как можно проверить что он работае, 80 порт прокидывается. И наш софт крутится в контейнере.

Теперь если зайти на IP компьютера:

user@Ubuntu:~/hello$ **ip a**

user@Ubuntu:~/hello$ **curl** [**http://10.0.10.200**](http://10.0.10.200)

Мы увидим наше веб приложение.

Строчка для запуска может быть использована для запуска контейнера на других операционных системах:

user@Ubuntu:~/hello**$ docker container run --publish 80:80 --detach almsys/hello:1.0**

Для этого нам нужен докер хаб. Создаем на нем логин almsys и репозиторий hello.

@Ubuntu:~/hello$ **docker login --username almsys**

**Password:**

Заходим и говорим чтобы наш собранный контейнер уехал в докер хаб:

**user@Ubuntu:~/hello$** docker push almsys/hello:1.0

Поскольку в докер хабе есть другие контейнеры, ему реально туда нужно закатить всего лишь маленький кусочик – наши изменения.

Главная прелесть, после закачки контейнера в докер хаб, мы можем запускать наш контейнер на любом компьютере, хоть на маке, линукс, фрибсдвне нашей сети.

Для этого на любом компьютере вводим, после чего нам приезжает образ и запускается:

**docker container run --publish 80:80 --detach almsys/hello:1.0**

И заходим на 127.0.0.1

Удобно ли это – да, можно ли это делать на Windows – да.

Мы укоротили мостик между продакшеном и релизом.

Но теперь нам не удобно вводить длинную строчку для запуска контейнера. Чтобы облегчить это, мы делаем установку Docker Compose.

Docker Compose – Что такое докер компуос это объяснение сколько и каких контейнеров нужно запустить чтобы заработало наше приложение.

Мы создаем файлик docker-compose.yml следующего содержимого:

**version: "3"**

**services:**

**hello:**

**image: almsys/hello:1.0**

**ports:**

**- "80:80"**

**restart: always**

**Теперь запускаем наш контейнер:**

user@Ubuntu:~/hello$ **docker-compose up**

Заходим на IP – 10.0.10.200 и видим наше приложение.

Теперь добавляем наш докер компоуз на гит:

user@Ubuntu:~/hello$ **git status**

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

docker-compose.yml

nothing added to commit but untracked files present (use "git add" to track)

user@Ubuntu:~/hello$ **git add docker-compose.yml**

user@Ubuntu:~/hello$ **git commit -m 'docker-compose'**

[master a18bae3] docker-compose

1 file changed, 8 insertions(+)

create mode 100644 docker-compose.yml

user@Ubuntu:~/hello$ **git push origin master**

**Что мы увидели?**

* **Удобно использовать уже готовые образы и стоится поверх**
* **Удобно делится работой и запускаться на разных системах**
* **Мы вносим только свои изменения, это очень быстро и просто**
* **Стало еще проще запускать контейнеры**

**А что дальше?**

* **А что, если мы хотим больше отказоустойчивости?**
* **Добро пожаловать в Kubernetes**

На своей машине запустим кластер minicube, перед этим завершаем работу контейнера:

user@Ubuntu:~$ **docker ps**

user@Ubuntu:~$ **docker stop <id container>**

Kubernetes управляется при помощи yaml файлов.

Устанавливаем minikube на Ubuntu:

user@Ubuntu:~/hello$ **curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 \**

&& chmod +x minikube

user@Ubuntu:~/hello$ **sudo mkdir -p /usr/local/bin/**

user@Ubuntu:~/hello$ **sudo install minikube /usr/local/bin/**

Ставим kubectl:

user@Ubuntu:~$ **sudo snap install kubectl --classic**

После установки minikube запускаем его:

user@Ubuntu:~$ **minikube start**

Проверяем как запустился Kubernetes:

user@Ubuntu:~/hello$ **kubectl cluster-info**

Создаем файл конфигурации для Kubernetes чтобы он нам создал заданный кластер:

user@Ubuntu:~/hello$ **vi k8s.yml**

---

apiVersion: v1

kind: Service

metadata:

name: hello-service

spec:

selector:

app: hello

ports:

- protocol: "TCP"

port: 80

targetPort: 80

type: LoadBalancer

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello

spec:

selector:

matchLabels:

app: hello

replicas: 4

template:

metadata:

labels:

app: hello

spec:

containers:

- name: hello

image: maniaque/hello:1.0

imagePullPolicy: Always

ports:

- containerPort: 80

Проверяем как запустились наши воркеры в количестве 4 штук:

user@Ubuntu:~/hello$ **kubectl get pods**

NAME READY STATUS RESTARTS AGE

hello-848f797fb5-54c7j 0/1 ContainerCreating 0 13s

hello-848f797fb5-crqwh 0/1 ContainerCreating 0 13s

hello-848f797fb5-rjt7x 0/1 ContainerCreating 0 13s

hello-848f797fb5-sxb4x 0/1 ContainerCreating 0 13s

Теперь проверяем наше приложение как оно запустились в Kubernetes с доступом по IP 192.168.49.2:

user@Ubuntu:~/hello**$ kubectl cluster-info**

Kubernetes control plane is running at https://192.168.49.2:8443

CoreDNS is running at https://192.168.49.2:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

Теперь узнаем на каком порту оно висит:

user@Ubuntu:~/hello$ **kubectl get svc**

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

hello-service LoadBalancer 10.101.216.81 <pending> 80:31820/TCP 11m

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 22m

user@Ubuntu:~/hello$ **curl http://192.168.49.2:31820**

Hello! 2024-10-30 05:21:01.733106

Или:

user@Ubuntu:~$ **watch curl** [**http://192.168.49.2:31820**](http://192.168.49.2:31820)

Если после перезагрузки не работает curl, то значить не запущен minikube (minikube start)

Чтобы он запускался автоматически, необходимо создать system unit.